Corpus Data Processing with Lexa

Raymond Hickey, University of Munich

Abstract

The present article offers an introduction to the software system *Lexa* which has been designed to facilitate the processing of corpus data. The main applications of the system, such as lexical analysis or information retrieval, are discussed with typical cases being examined. After a brief explanation of what files types can be handled by the *Lexa* suite the question of text categorization is looked at. Then a detailed presentation of automatic tagging is offered. Particular attention is given to the degree to which such operations can be customized to users' needs along with the transfer of textual data to a database environment for the purpose of constructing lexical databases. The article concludes with a selection of further applications of the programme suite in the general field of corpus data processing.

1.1 Purpose and scope

The purpose of the present article is to introduce the software suite *Lexa* to the community of linguists interested in learning about software for the management and processing of text corpora on a personal computer. The system *Lexa* is a complete text retrieval system with its major emphasis in the general area of corpus processing, particularly the tagging and analysis of texts and the derivation of lexical databases from such texts and their subsequent handling with appropriate database management software. Given the scope of this introductory article only a brief sketch of some typical applications of the software can be offered. I have chosen to look more closely at the area of lexical and grammatical analysis of texts and to follow this with some references to databases, information retrieval and the processing of historical corpus texts. Hopefully the descriptions below will convey to readers (and potential users) an impression of what is the aim and scope of the *Lexa* suite is.

1.2 Availability of Lexa

The present suite of programmes consists of more than 60 executable files comprising some 4MB along with additional sample data. The set is self-installing and requires no particular hardware apart from a fixed disk with at least 5MB of free space and of course additional space for any primary corpus data which users may wish to process. Accompanying the software are 3 volumes (each between 250 and 300 pages in length) which contain both extensive documentation of the programmes and exemplary discussions of typical processing tasks. The volumes refer to typical data processing areas covered by the software, namely (i) *lexical analysis and information retrieval*, (ii) *database and corpus management* and (iii) *general file management*. The texts are intended to be suitable for beginners and include comprehensive glossaries of all technical terms used in the body of each volume. The programmes and texts have been published by the University of Bergen and are available from the Norwegian Computer Centre for the Humanities

in Bergen as of Spring 1993. As the software is intended for bona fide linguistic research there is no special copyright agreement concerning its use nor is there any kind of programme protection.

1.3 Making use of Lexa

For computer users who are acquainted with the basics of personal computing the use of the *Lexa* suite should present no difficulties. It is organized as a collection of over 60 programmes. Of these some are major and other are minor. To start with the set can be surveyed by means of a so-called control centre. This is a programme which offers the user a brief summary of each member of the suite and allows him or her to load any programme, automatically returning to the control centre for renewed starting of a further programme. By these means the user can very quickly ascertain what the individual programmes of the suite actually do. An alternative launching pad for all programmes is available as a desktop which complies in its design to the SAA (*system application architecture*) standard of IBM which users will be familiar with from such environments as *Microsoft Windows*. Indeed all major programmes employ a system of picklists available on an entry level to the particular programme, allowing the user to activate any option of the programme by simply moving a highlight bar and pressing the Return key. Again for all major programmes, online help and mouse support are included.

Furthermore configuration information is stored to disk and can be used during a later work session. As a matter of principle all the main programmes can be run interactively or in the so-called *batch mode* in which a programme loads itself, gleans it configuration information from a setup file, executes and returns the user to DOS automatically without it being necessary to supply user input during the execution of the programme. The advantage of this is that various tasks can be executed automatically as a group without the user necessarily being present. The time factor involved in complicated and intricate processing tasks then becomes irrelevant. All programmes which collect information about texts or databases during their operation can write this to an output file (for later inspection with a text editor such as that supplied with the *Lexa* suite) apart from displaying information collected on the screen. Note that all input files for processing must be either ASCII texts or databases in the *dBASE* format for the *Lexa* programmes to accept them as valid input. This is not a restriction but rather a gain in flexibility over word processor files (such as those generated by *WordPerfect* or *Microsoft Word*) as the source files can come from any computer environment, not just a personal computer, e.g. from a mainframe or a Unix work station.

2 Corpus data processing

It should be mentioned at the outset of this section that the *Lexa* suite was designed to be used with any text corpus. The programmes makes no assumptions about the source of input texts apart from their being pure ASCII texts. Nonetheless users will notice that many references are made both within the documentation and with the software to the Helsinki corpus of diachronic English (Kytö, 1991). There are definite reasons for this which have to do with the association of the present author with colleagues in the Department of English in Helsinki, notably with Matti

Rissanen and Merja Kytö both of whom have been instrumental in realizing the Helsinki corpus (Kytö and Rissanen, 1992:7ff.). I would be pleased to be mentioned in connection with the latter corpus and for my software to be used with it for data processing tasks. At this stage my only desire is to emphasize that the *Lexa* suite can be applied to any corpus, including the corpus of Irish English being presently compiled by the present author or already available corpora such as the Lancaster-Oslo-Bergen corpus.

2.1 Categorization of texts

All the programmes of the *Lexa* suite which process data can take as their input text files which are specified by the user. There are a variety of means for specifying such files. The easiest of all is for the user to select a file from a directory listing presented on the desktop of one of the data processing programmes. Another means is for users to enter a file template which encompasses the files to be affected by an operation to be performed. Such means are mechanical and depend entirely on file grouping according to the names used by the operating system. A more flexible system is available for all the programmes which perform information retrieval tasks. Here users can specify that a programme use as its input those files which are deposited in a so-called *list file*. The latter is a small ASCII file which consists of several file names each on a separate line of the file. There need be no similarity in name between the files listed, this freeing one from the straightjacket of file names on the operating system level. The scope of this option is greatly increased if one considers carefully how such list files can be generated. To begin this discussion allow me to present briefly what is known as a file header and the widespread format used for this, the *Cocoa file header format*.

Among the text corpora available today many make use of a format for including information relating to the contents of files. A commonly used format is that called the Cocoa format which consists of a series of parameters which characterize the text in question.

1:<B = 'name of text file'>	2:<Q = 'text identifier'>
3:<N = 'name of text'>	4:<A = 'author'>
5:<C = 'part of corpus'>	6:<O = 'date of original'>
7:<M = 'date of manuscript'>	8:<K = 'contemporaneity'>
9:<D = 'dialect'>	10:<V = 'verse' or 'prose'>
11:<T = 'text type'>	12:<G = 'relation to foreign original'>
13:<F = 'foreign original'>	14:<W = 'relation to spoken language'>
15:<X = 'sex of author'>	16:<Y = 'age of author'>
17:<H = 'social rank of author'>	18:<U = 'audience description'>
19:<E = 'participant relation'>	20:<J = 'interaction'>
21:<I = 'setting'>	22:<Z = 'prototypical text category'>
23:<S = 'sample'>	24:<P = 'page'>
25:<L = 'line'>	26:<R = 'record'>

This information can be accessed by the information retrieval software of the *Lexa* suite in the following way. A programme (called *Cocoa*) extracts the header information from any set of input files and deposits this in a database. Then with the database manager *DbStat* one can load the database just created and impose a filter

on it by which only those records remain visible which meet a certain condition. Assuming that one generates a database of the Cocoa header information in the files of the Helsinki corpus then one could specify a filter to which only those records (i.e. file headers) correspond which represent translations (Item 13) of Middle English (Item 6) prose (Item 10) texts. A list of the files for which this header information obtains can be generated by creating a list from the field information for Item 1 (*name of text file*). The list file created by these steps can in its turn be used as the source of the file names for an information retrieval operation with other parts of the *Lexa* suite so that only Middle English prose translations from the corpus are examined. In addition the user can specify with the retrieval programmes from the set (such as *Lexa Pat* and *Lexa Context*) that the Cocoa information of the files examined be enclosed in the output file of statistics generated during a search.

The example just given is typical inasmuch as it illustrates how different parts of the *Lexa* suite link up together. For any prospective users of the programme package it is essential to grasp the interrelationships between items of software. A disconcerting and sadly not uncommon experience of the present author is that users complain that some feature is not present when in fact it is but they have not realised it as they fail to grasp the potential of certain programmes.

2.2 Lexical and grammatical analysis of texts

The following section is intended to convey an impression of what tasks can be accomplished by using the main programme of the *Lexa* suite. To begin with a word of explanation regarding nomenclature is necessary. In the *Lexa* suite the main programme for carrying out lexical and grammatical analysis is itself termed *Lexa*. All other programmes consist of *Lexa* and a further word which refers to what function they perform, thus the pattern matcher is called *Lexa Pat*, the programme for locating syntactic contexts is called *Lexa Context*, etc. The names of these files on the operating system level consist of the function word (or an abbreviation of this) preceded by the letter 'l', e.g. lpat, lcontext, etc.

2.2.0 Preamble: What is meant by 'text'

It is fair to say that any data which users of the *Lexa* suite will process will initially be in text form. This is due to the fact that the software is intended for use with text corpora. A text corpus consists of a collection of files which contain pure text, i.e. the files it comprises are in the so-called ASCII format. An exception to this is the special case where one commences with texts which have been indexed for use with particular software in advance (this is the case of the Helsinki corpus which is available on CD-ROM in a pre-indexed form for use with the commercial text retrieval system *WordCruncher*. But even in such instances, the actual text files usually remain in the original ASCII format, i.e. they do not contain any information which is specific to a certain word processor. This is in sharp contrast to the situation with the text files one may generate with one's word processor on a personal computer. Here the file which contains a text will also include information for the formatted output of the text on a printer, e.g. information concerning the layout of the page (page length, left/right margins, etc.), the attributes used for certain letters or words (boldface, italics, etc.). Formatting information is always

specific to a particular word processor and so cannot be intelligently interpreted by some other programme. To analyse texts with *Lexa*, which have been processed or created with a word processor, these must be stored to disk without any formatting information (this option will always be available with one's word processing software).

Users of computers should thus bear in mind that in computing the term *text* has a very definite meaning. A text is a collection of informational units (usually bytes) which are arranged as an unstructured number of lines. There may well be a semantic structure to the text (determined by its contents) but for the computer a text contains no inherent structure. In computing, the term 'text' is frequently used somewhat loosely to refer to an ASCII (i.e. non-formatted) text.

With an ASCII text there is a pre-defined set of characters which trigger the end of a line: ASCII $13 and ASCII $10 (called *carriage return* and *line feed* respectively). Any programme processing an ASCII text thus knows where a line comes to an end.

2.2.1 Tagging a text corpus

Before any kind of useful lexical and grammatical analysis can be performed on a text corpus it is necessary for it to be tagged. This is a task which may well have been carried out in advance by the compilers/distributors of the corpus. However it is not always the case. For instance the Lancaster-Oslo-Bergen corpus is available in a pre-tagged form whereas the Helsinki corpus is not. Thus those users of the latter who wish to tag it (to what extent is a secondary matter) will require software such as *Lexa*. Note that the tagging scheme used for the Lancaster-Oslo-Bergen corpus can be applied to the Helsinki corpus in either the original or a user-defined form (the decisions on what forms in a text are to obtain what grammatical tag from a set of tags are made by the user; the details of this procedure are outlined below). In essence tagging works as follows: each word in a file is examined and a label is added to it to identify it grammatically.

Stretch of text before tagging

A marchant was ther, with a forked berd,
In motlee, and hye on horse he sat,
Upon his heed a Flaundryssh bevere hat,
His bootes chasped faire and fetisly

Stretch of text after tagging

A_ART marchant_NOUN was_VERB ther_PREP, with_ADV a_ART forked_ADJ berd_NOUN,
In_PREP motlee_NOUN, and_CONJ hye_ADJ on_PREP horse_NOUN the_PERPRO sat_VERB,
Upon_PREP his_POSSPRO heed_NOUN a_ART Flaundryssh_ADJ bevere_NOUN hat_NOUN,
His_POSSPRO bootes_NOUN chasped_VERB faire_ADJ and_CONJ fetisly_ADV

It is obvious from the above illustration that the tags are placed after the words they refer to and are separated by a single underscore (_, the character used can be specified by the user). This is a widely accepted convention (cf. the London- Lund or the Lancaster-Oslo-Bergen corpora). The tag itself is the capitalised abbreviation used to unambiguously classify the word in question. Needless to say for an ensuing grammatical analysis of any sophistication, it would be necessary to devise more refined categories than those used for illustrative purposes above.

Tagging may be done manually by the computer operator deciding as he or she goes through the text how each word is to be classified. However, the task is impracticable unless one resorts at least to a degree of automatic tagging. Both methods are available with *Lexa* as is a combination of the two.

2.2.2 Lemmatisation

The term *lemma* is used in lexical data processing with the equivalent meaning of *lexeme* in general linguistics. A lemma is thus an abstraction of the set of inflected forms which are united by a common semantic core. For instance the attested forms *walk, walks, walked* and *walking* all belong to the lemma WALK. In the lexical analysis of a corpus the concern is then to group together inflected forms and somehow mark them so that their semantic affiliation is obvious.

2.2.3 Automatic tagging

Lexical tagging. With any type of computer analysis involving automatic procedures it is necessary for the computer operator to initially lay down the criteria which the system is to use for classification.

When tagging a corpus with *Lexa* this can be achieved as follows. The computer operator creates a file with a list of tags contained in it. After each tag are listed those forms which can be given the tag in question. When involved in lexical tagging, the user enters the keyword *Lemma* on a line and after this the lexeme to which the ensuing form belongs.

```
LEMMA: SING
TOKENS: WORDS
sing
sings
singing
sang
LEMMA: WALK
TOKENS: WORDS
walk
walks, etc
```

Before initiating a tagging session the computer operator specifies which tag list file is to be used for the run. The system reads the file and fills an internal table with the definitions contained in the tag list file.

Technically the steps are as follows: a two-dimensional array is allocated in memory from the heap (that section of system memory which can be used for data

by the programme which is currently running). One dimension of the array contains the names of the tags which are defined in the tag list file; the second dimension contains the forms which are defined as being tokens of a particular tag. You can envisage this as a series of rows and columns with types and tokens occupying vertical and horizontal positions respectively. For every word in a text which is tagged, *Lexa* combs through the entire array of tags to see if the current word is a token of some tag or other. If the search for a match is positive the relevant tag is attached to the current word and the system proceeds to the next word.

For the *Lexa* programme a tag can be of two basic types: (i) it refers to a lemma, i.e. a dictionary entry which subsumes a whole series of inflected forms, in which case the tag begins with the keyword LEMMA or (ii) it indicates a word class or morphological category, in which case the keyword CLASS is to be found after the tag. As can be seen from the above example, on the next line after the lemma the keyword TOKENS occurs; immediately following this is either the word WORDS or STRINGS. This is noted by *Lexa* and when lemmatising a text the tokens which are found in the input file are then either treated as whole words or as strings. Take an example to see what advantage is to be gained from this. Say you have defined a set of prepositions as follows:

```
CLASS: PREP
TOKENS: WORDS
for
in
out
on
```

When later combing through a text *Lexa* will only mark occurrences of these tokens as instantiations of the word class PREP (the same would apply to a lemma) which form whole words thus avoiding incorrect tagging such as foreigner_PREP, intake_PREP, outgoing_PREP, button_PREP.

Grammatical tagging When dealing with inflected forms it is no longer sufficient to use list of lexemes as a basis for successful tagging. The solution is to create a tag list file which consists of sub-word morphemes and to allow the data processing software to determine class affiliation on the basis of a morpheme being present in a word form or not. Consider the following extract from a list file for grammatical tagging of a group of Middle English texts:

```
CLASS: PAST_PART
TOKENS: STRINGS
#y*e#
#y*en#
CLASS: ADV
TOKENS: STRINGS
ly#
CLASS: PREFIXVERB
TOKENS: STRINGS
#pre
#fore        CLASS: FRENCHVERB
TOKENS: STRINGS
```

ceive#

It should be obvious just what type of returns one is expecting with such a list. Note that the *fore* tag for a prefix verb will not of course yield unambiguous results as words like *forehead, forelimb* would be returned if present in a text which is examined. Equally the ending *ly* will return words like *fly* which must be re-classified manually afterwards.

One solution to the difficulty of unacceptable returns is to perform some other type of tagging beforehand which would capture these forms. Once they are tagged they will not be re-tagged by the system.

Another solution would be to create a stop word file (see below) with those forms listed in it which one does not want tagged. Of course this alternative is really only viable if the set of potentially undesirable tags is fairly small.

With the *Lexa* programme if the symbol '#' (or a user-specified word delimiter) is placed before an affix then it must occur at the beginning of a word; if it is placed at the end then it must be at the end of a word. The word delimiter can be used at the beginning *and* end of a token. DOS wild cards, * and ?, can also be used to leave (one or several) characters unspecified.

2.2.4 Cumulative tagging

Not all the words of a text need to be tagged on one run. In fact it is sensible to tag the most obvious words (those which constitute a small closed class of items) first and then gradually work on to the more difficult classes with hopefully only a small number of non-classified items left which have to be dealt with manually by the computer operator at the end.

When the data processing software examines a text a mechanism can be used to determine whether any given word which it strikes upon has already been tagged or not. If every tag begins with a pre-defined character, say an underscore, and if the underscore does not occur as a constituent of any normal word of the text then any given word can be examined to see if it contains an underscore. Should this be the case then the word has been tagged on a previous run, if not, then the data processing software is to attempt tagging this time.

2.2.5 Manual tagging

No matter how good the tagging algorithm is there will always be a residue of word forms which cannot be automatically classified. These must be tagged manually. To do this the programme must demand that the computer operator decide on the tag to be attached to any words found in a text being examined which have not yet been classified. Bear in mind that manual tagging is always necessary with ambiguous forms as the data processing software can only use formal criteria to determine class affiliation. Within *Lexa* there is a text editing level with special features which refer to tagging. The text currently loaded in memory can be viewed and edited at any stage (from the desktop). When editing a text, manual tagging can also be carried out by means of a number of inbuilt macro facilities.

2.2.6 Stop words

The easiest forms to tag are those which form a small closed class, e.g. the articles in English. However, these forms are usually of little interest to the linguist examining a corpus. What is then desirable is to filter them out and concentrate on the remaining forms. This can be achieved quite easily. The first step to this end is to create a list of those words (called *stop words* in computer jargon) which are to be ignored. When the data processing software then examines a text file it first checks to see if a given form has been labelled as a stop word (by looking it up in an internal table). If so the form is ignored and it precedes to the next.

Evading stop words can be achieved either by excluding them from a tagging operation or by erasing them from a file to start with. One might care to create a temporary version of a text file without stop words as this would speed up tagging later (after all there would then be no cases in which forms are examined and then discarded by the system).

2.2.7 Locating and altering tagged forms

At any point when processing a corpus it may be expedient to both tag certain sets of forms and then locate them to see just what words were affected by the tagging. This can be realized within *Lexa* when dealing with single files. If a broader scope is required, covering a group of files, for instance, then the easiest way of satisfactorily locating tagged forms involves using one of the many information retrieval programmes in the *Lexa* set. The supplementary programmes one can avail of are *Lexa Pat*, *Lexa Search* or *Lexa Context*, for text files, or *Lexa DbPat*, for databases.

It may well occur that once one has tagged a text or texts one wishes to alter the tagging done. There is a general utility *Lexa Sweep* which can be used among other things for this purpose. One specifies the form of an old tag, that of the new one to replace it and the set of files to be affected by the operation. One can also use *Lexa Sweep* to remove tagging, i.e. one says what tag is to be located and leaves the replace string empty. This removes a tag without inserting a substitute in its place.

2.2.8 Multiple tag files and input texts

When *Lexa* is run in the so-called *interactive mode* the user chooses an input text from a directory listing offered on the *Lexa* desktop. By choosing a further option from the relevant picklist one can then proceed to tag the text chosen. This procedure is sensible when one is getting acquainted with computerized tagging and the functioning of the programme *Lexa*. However, with time one will wish to be more flexible in data processing. To achieve this *Lexa* must be executed in the *batch mode*. By this is meant that all the information necessary for the operation is specified in an initialization file. The programme then derives the values for all its user-specifiable parameters by examining this file on loading. One can demand that *Lexa* analyse a series of texts by using a file template (a specification with one or both of the MS-DOS wild cards * and ?) instead of an explicit file name as input text for analysis. *Lexa* will then examine any files found in the data directory which match this template. The same technique can be applied when specifying the name

of the tag list file to be used. Should a file template be entered at this point in the initialization file then *Lexa* will attempt tagging each file of the input text template with tag definitions from each of the files in the tag file list template.

During batch mode operation *Lexa* informs you of what it is doing. However, no user input is necessary so that the presence of the user is not required. Furthermore very large files can be processed in the batch mode. Should these not fit into available system memory then *Lexa* can use the so-called *file-slice mode* in which it loads a section of the text currently being examined and when finished proceeds to the next section until the file has been analysed completely.

2.3 Constructing lexical databases

A frequent desideratum when lexically analysing a corpus is to construct a dictionary with grammatical information included on the word forms which constitute the dictionary. Such a task becomes quite easy with a lemmatised text. The first step (or rather sequence of steps) is the complete lemmatisation of the texts in question. Once this has been achieved the data processing software can now extract information from the text and deposit it in a database. Recall that a database is a structured file which consists of a number of records, each in turn consisting of a number of fields. A non-electronic parallel would be a box of index cards. Each card corresponds to a record and assuming that there are ordered divisions on each card then these would represent the equivalents of record fields. A typical lexical database has one record per word form.

The programme *Lexa* constructs a (primary) lexical database by generating an empty database with a minimum of four fields as follows (this is all that is required at this stage; lexical databases can of course be manipulated later with the database manager of the *Lexa* suite, *DbStat*).

Field 1: TOKEN
Field 2: LEMMA
Field 3: FREQUENCY
Field 4: REVERSE

Each word form in the database occupies a record of its own. The system starts by checking to see if a particular record is already present in the database. If not, a new record is appended and the word form is entered automatically in the field TOKEN. The lemma is extracted from the tagged word by locating the lemma divider character (by default an underscore) and copying the remainder of the word form (up to the next space or item of punctuation) into the field LEMMA. The field FREQUENCY is incremented each time an occurrence of the particular type of that record is found. Lastly, the field REVERSE contains the word form in reverse order. The idea behind this is to allow users to create a reverse dictionary (by sorting the database on the field REVERSE) thus making it much easier to recognize what inflectional information is contained in the word forms of the database.

After the process has been completed, you are left with a database which has as many records as there are unique word forms in the corpus examined. Note that should a word form not be lemmatised in the corpus for some reason then the form is nonetheless added while the field LEMMA is left empty.

Apart from the database type just outlined above it is possible with *Lexa* to

generate a database which has one record per lemma. This is a secondary database which is realised by first creating an empty database manually (e.g. by deriving a shell database from a *Lexa* database via the appropriate option on the *DbStat* desktop) and then importing the information from a *frequency list file* into it subsequently. The information in the latter type of text file (which is generated by an appropriate option in *Lexa*) is organized into lines with three items on each: a *unique* word form, the lemma attached to it and the number of times it occurs in the database (frequency). These items of information always begin in the 1st., 33rd. and 49th. columns of each line in the text file respectively. Due to this organization it is easy to import the information into a database by treating the frequency text file as an SDF (= *system data format*) file and using it as the source of a text importation operation with a database manager (such as *DbStat*). The databases generated by *Lexa* are always in the *dBASE* format. This is by far and away the most commonly available and readable format on personal computers. The lexical databases outputted can be read by virtually any other database manager apart from the one supplied with the *Lexa* suite.

2.3.1 Generating database-readable text files

In order to move the data of a text file to a database environment it is essential to either create a database or a file which can be read directly by a database. The latter course of action is covered by an option within *Lexa*. It generates a so- called delimited text file from the text in memory. Using a specially reserved character as a delimiter of certain contents on each line, an output text is created which can be read by a database manager and which leads to the information in the text file being properly assigned to the fields of a database.

2.3.2 Merging textual information with databases

As a corpus will in all probability consist of a number of text files, generating a database from the word forms of an entire corpus will require that the data from each text file be transferred to a database. However, it would be pointless to create a new database each time a text is analysed. Instead what one needs is an option in which data is added in a cumulative fashion to a single database so that it reflects the lexical structure of more than one input file.

This is realised with a further option within *Lexa*. For the first text to be analysed one creates a database with the *Generate database* option. With all subsequent files one merges databases. On doing so one must first of all choose a database to merge textual data with. Care should be exercised here that the database chosen is one which was generated by *Lexa* at some previous stage. If not, *Lexa* issues an error message and refuses to continue.

Assuming that the database is acceptable to *Lexa*, it now combs through it and undertakes one of two steps: (i) adds the word form in the current text file in memory to the external database if this form is not already contained in the latter, (ii) increments the frequency field of the database should the current word form from the memory text already occur in the external database.

2.3.3 Statistical operations and databases

The database manager of the *Lexa* suite is especially geared towards the processing of numerical data. To this end it contains a wide range of statistical options. These can be applied to the frequency figures generated by many other programmes such as the main programme *Lexa*. All such programmes can posit the result of some operation which generates frequency tables in a text file of a special kind which can then be read into the field of a set of records with the database manager.

The statistical options available with *DbStat* fall into three main groups.

(i) Options for preparing or arranging data.
(ii) Options for determining central tendency.
(iii) Tests which involve (two) sets of data.

The first group will perform such tasks as ranking data, sorting them, generating interval and frequency lists or displaying the range in a set of data. In this case as with others connected with calculations with *DbStat* a set of numerical data is defined by the entries in a numerical field for the records of a database.

With the second group one has a series of options which determine central tendency with a set of data. Examples of these are median, mode, interquartile, variance, standard deviation (biased and unbiased) apart from simpler types of calculations.

The purpose of the third set is to carry out operations which are particularly suited to the type of non-parametric data found in linguistic material. Note that for inferential statistics two sets of data are always required. Three possible relationships may obtain between these.

(i) One set may represent a set of expected values and the other a set of observed values. (Chi-square)

(ii) The two sets are possibly correlated. (Pearson, Spearman)

(iii) One set may be a sample and the other the parent population from which the sample is putatively drawn. (Mann-Whitney, Wilcoxon, Sign-test, F-test)

The types of test available in *DbStat* for the particular set of data are indicated in brackets above. It would go far beyond the scope of the present introductory article to explain and illustrate the statistical options which are put at the user's disposal with the database manager *DbStat*. It must suffice it at this point to hint at them; I cannot do anything else but refer the interested reader to the documentation accompanying the software which contains greater detail on the use of such options.

2.4 Generating concordances

A further set of features in *Lexa* is concerned with the generation of text files in which word forms are highlighted in order to easily recognize the context in which they occur. These are traditional types of files to be found with concordance programmes and are included at this point to offer similar facilities to users of *Lexa*.

Concordance file (i) This option generates a so-called KWIC concordance. The abbreviation stands for "key word in context" and as the name implies, each occurrence of unique word forms is highlighted (by spacing on either side) in the context in which it is to be found.

Concordance file (ii) The second type of concordance file has similarities with what is known as a KWOC file from the designation "key word out of context". The keyword is however not so much out of context as not centred in the line in which it occurs. This type of file simply contains the tokens of word types enclosed in curly brackets for easy recognition.

Concordance files normally contain all the unique forms found in the text file currently in memory. However, if you set the relevant option in the initialization file to 'on', you can force *Lexa* to create a concordance file with only a selection of word forms. These are contained in a text file which is also specified in the initialization file. A word list for concordance generation consists of a number of words each occupying exactly one line in the input text file. This option can be tested with the supplied file excerpt.frm.

2.5 Lexical density

Token lexical density A text file is created with the present option which contains the unique word forms of any text arranged in ascending alphabetical order of their frequency, offering the user a picture of the density of word forms in the text. *Lemma lexical density* This is similar to the previous option with the difference that the lemmas of the word forms (i.e. the tags) in a chosen text are arranged in the output text file according to frequency of occurrence.

3 Information retrieval with Lexa

One very large area which has only been touched on indirectly so far concerns information retrieval. By this is meant the selective extraction of user-specified information from the texts of a corpus. Note that *Lexa* can handle such tasks with both texts (the normal form of a corpus) and databases (a derived form).

The main retrieval programme is *Lexa Pat*. Its basic function is to locate user-specified strings in text(s), writing the results of a search along with statistics gathered during such a search in a text file. The programme contains a number of additional extras to improve flexibility. For one thing sets of files can be combed through. To indicate this one can use a DOS file template or a list file as discussed above (see 2.1.) which can be generated by means of the programme *Cocoa* thus restricting searches to a (user-specified) subset of files. Furthermore the forms searched for can be indicated by a normal file template or by the user conveying the name of a list file in which a series of words which are to be searched for are included. The forms in such a list file may in their turn also include DOS wild cards to broaden the base of possible matches which might be returned by the system.

As with the major programmes of the suite, *Lexa Pat* is configurable, writes all the information which it collects during its operation to a text file and most

importantly can be run in the so-called batch mode (again, see above 1.3.).

3.1 Locating syntactic contexts

The information retrieval software of the *Lexa* suite is not confined to the location of single word forms. Very often the linguist will be interested in finding syntactic contexts. The programme *Lexa Context* is intended to fulfill this need. Basically what the programme will do is to look for any word or string and then locate a second word or string within a specified number of words or characters thus returning a syntactic context. The only requirement for the programme is that the context be formally specifiable. The user can use the DOS wild cards ? (for one unspecified character) and * (for more than one such character) in the words and/or strings used in a search, e.g. locate contexts within the following frame: *that*, up to 8 intervening words, **ed*. This would in all probability return contexts of relative clauses which end in past forms of verbs. Of course the reliability of the returns depends on how well the context can be and is in fact specified by the user. Nonetheless, few contexts will be entirely unambiguous. A solution to this quandary is to allow the user to decide whether the context returned by the programme represents a genuine find for the context the user is looking for. You can force *Lexa Context* to display each context on screen and to ask the user whether it is genuine or not. By these means the user can decide what contexts are acceptable and hence to be added to the statistics which are collected during a search.

This programme has been used effectively by the present author to look at the syntactically deviant forms in the dramas of John Millington Synge (part of the corpus of Irish English under preparation). It was successfully employed to locate structures like *after* + present participle as the indicator of a perfective aspect in Irish English and the use of *for to* + infinitive in clauses of intention as well as general fronting with cleft sentences introduced by *it is* and a topicalized element from a sentence. Note that *Lexa Context* can take the sentence as its primary unit of investigation. The user conveys to the programme what items of punctuation signal the end of a sentence. Going on these it divides the text it examines into sentences and returns statistics which refer to this organizational unit.

4 Additional facilities

4.1 Normalization of texts

An editorial task which arises quite frequently is the normalization of texts. There are a variety of reasons why this should be so. A common one is to reduce the distracting effect which irrelevant data can have on users analysing a text or set of texts. Such normalization might involve the levelling of irregular verb forms with a text which one is investigating for some other information than verb composition. This task can be achieved easily with a utility in the *Lexa* suite called *DbTrans*. Essentially what it does is to examine an input text or texts and going on a dictionary database which is conveyed to it by the user carries out a series of substitutions. In the hypothetical example just quoted, the user would specify what variant verb forms are to be regarded as manifestations of what normalized forms. The programme then consults the database specified and if it locates a form in the

input field of the database replaces this by that in the output field. The net result is a group of replacements which, if the substitutions are correctly specified by the user in the database consulted, leads to normalized output text.

4.2 Display of older texts from the Helsinki corpus

In the compilation of the Helsinki corpus its designers made a wise decision to encode special symbols which are necessary for Old and Middle English by using so-called 'escape sequences' (Kytö and Rissanen, 1988). These are sets of two bytes, the first of which is a reserved character which indicates that the following one is not to be taken at its face value but as a special symbol which cannot be represented using the IBM extended ASCII character set to be found by default on all personal computers, in fact the Helsinki corpus gets by with characters from the lower area of this set. So for instance the 'eth' character of Old English (a crossed 'd' which along with thorn, a Runic character, was used to represent the inter-dental fricatives of this stage of the language) is encoded as '+d'. Thorn itself is indicated as '+t', the medieval form of *g* 'yogh' is encoded as '+g', etc. The advantage of such a coding scheme is that of portability: texts can be transferred effortlessly from one environment to another, e.g. from a personal computer to a mainframe or a work station, from one operating system to another without entailing loss of data. The disadvantage should be obvious: one cannot see Old and Middle English symbols as they would be represented in printed form. For the linguist involved in analysis of medieval texts this is untenable in the long term. To alleviate the situation a programme has been included in the *Lexa* set which will convert all escape sequences used for the Helsinki corpus into single characters. If one then uses the special Old English character set supplied with the *Lexa* suite then one actually sees Old and Middle special symbols as they appear in the printed forms of medieval texts. Furthermore one can reverse the conversion of Helsinki texts thus allowing portability to another environment at any time. A special keyboard driver and a printer driver for *WordPerfect* as well as both dot matrix and laser printer fonts are supplied with *Lexa* which allow one to enter from the keyboard, view on the screen and output on paper the Old and Middle symbols required for the earlier texts of the Helsinki corpus.

4.3 A word on utilities

The third volume of the *Lexa* suite which is concerned with general file management bears the title *Utility Library*. It embraces a series of programmes which perform various housekeeping tasks necessary for efficient data management on a personal computer. As such the programmes are not primarily involved in corpus processing, but should nonetheless not be neglected by users. Mention should just be made here of the fact that CD-ROM drives are supported by the utility software which means that the ICAME CD-ROM of English Language Corpora which is available form the Norwegian Computing Centre for the Humanities can be managed directly by the *Lexa* software.

References

Aarts, Jan and Willem Meijs (eds.) 1984. *Corpus linguistics: Recent developments in the use of computer corpora in English language research*. Amsterdam: Rodopi.

Aarts, Jan and Willem Meijs (eds.) 1986. *Corpus linguistics II: New studies in the analysis and exploitation of computer corpora*. Amsterdam: Rodopi.

Aarts, Jan and Willem Meijs (eds.) 1990. *Theory and practice in corpus linguistics*. Amsterdam: Rodopi.

Aijmer, Karin and Bengt Altenberg (eds.) 1991. *English corpus linguistics: Studies in honour of Jan Svartvik*. London: Longman.

Akkerman, Eric, Willem Meijs H. and Voogt-van Zutphen, H. 1987. Grammatical tagging in ASCOT. In *Corpus linguistics and beyond: Proceedings of the Seventh International Conference on English Language Research on Computerized Corpora*, ed. by Willem Meijs. 181-193. Amsterdam: Rodopi.

Altenberg, Bengt 1991. A bibliography of publications relating to English computer corpora. In *English computer corpora: Selected papers and bibliography*, ed. by Stig Johansson and Anna-Brita Stenström. 355-396. Boston: Mouton de Gruyter.

Biber, Douglas 1988. *Variation across speech and writing*. Cambridge: University Press.

Butler, Charles 1985. *Statistics in linguistics*. Oxford: Blackwell.

Francis, W.Nelson 1980. A tagged corpus - problems and prospects. In *Studies in English linguistics - for Randolph Quirk*, ed. by Sidney Greenbaum et al. 192-209. London: Longman.

Garside, Roger 1987. The CLAWS word-tagging system. In *The computational analysis of English: A corpus-based approach*, ed. by Roger Garside et al. 30-41. London: Longman.

Garside, Roger and Geoffrey Leech 1982. Grammatical tagging of the LOB Corpus: general survey. In *Computer corpora in English language research*, ed. by Stig Johansson. 110-117. Bergen: Norwegian Computing Centre for the Humanities.

Hickey, Raymond 1993a. *Lexa. Corpus Processing Software, 3 Vols. Vol.1: Lexical Analysis. Vol.2: Database and Corpus Management. Vol.3: Utility Library* Bergen: Norwegian Computing Centre for the Humanities.

Hickey, Raymond 1993b. An assessment of language contact in the development of Irish English. In *Language contact and linguistic change*, ed. by Jacek Fisiak. Berlin: de Gruyter.

Hockey, Susan and Ian Marriott 1980. *Oxford Concordance Program: Users' manual*. Oxford: Oxford University Computing Service.

Johansson, Stig 1986. *The tagged LOB Corpus: User's manual*. Bergen: Norwegian Computing Centre for the Humanities.

Johansson, Stig and Anna-Brita Stenström (eds.). 1991. *English computer corpora: Selected papers and research guide* Berlin: Mouton de Gruyter.

Kytö, Merja 1991. *Manual to the diachronic part of the Helsinki corpus of English texts*. Helsinki: Department of English.

Kytö, Merja, Ossi Ihalainen, and Matti Rissanen (eds.) 1988. *Corpus linguistics hard and soft*. Amsterdam: Rodopi.

Kytö, Merja and Matti Rissanen 1988. The Helsinki Corpus of English Texts: Classifying and coding the diachronic part. In *Corpus linguistics*, ed. by

M.Kytö et al. 169-180. Amsterdam: Rodopi.

Kytö, Merja and Matti Rissanen 1992. A language in transition: The Helsinki Corpus of English texts. ICAME Journal 16: 7-27.

Lancashire, Ian. 1991. *The humanities computing yearbook 1989-90: A comprehensive guide to software and other resources* Oxford: Clarendon Press.

Leech, Geoffrey, Roger Garside and Eric Atwell 1983. The automatic grammatical tagging of the LOB Corpus. ICAME Journal 7: 13-33.

Meijs, Willem ed. 1987. *Corpus linguistics and beyond: Proceedings of the Seventh International Conference on English Language Research on Computerized Corpora*. Amsterdam: Rodopi.

Oostdijk, Nelleke 1988. A corpus linguistic approach to linguistic variation. *Literary and Linguistic Computing* 3/1: 12-25.

Svartvik, Jan 1987. Taking a new look at word class tags. In *Corpus linguistics and beyond: Proceedings of the Seventh International Conference on English Language Research on Computerized Corpora*, ed. by Willem Meijs. 33-43. Amsterdam: Rodopi.